Advantage-Devices

Modbus Protocol User Guide

Revision C January 3, 2006

Part Number GC-800-250

Advantage-Lab GmbH
Otto-Röhm-Strasse 69
64293 Darmstadt / Germany

www.advantage-devices.com
info@ advantage-lab.com

Copyright and Trademark

Copyright ♥ 2013, Advantage-Devices GmbH. All rights reserved.

No part of this manual may be reproduced or transmitted in any form for any purpose other than the purchaser's personal use, without the express written permission of Advantage-Devices GmbH. Advantage-Devices GmbH has made every effort to provide complete details about the product in this manual, but makes no warranty of any kind with regard to this material, including, but not limited to, the implied warranties of merchantability or fitness for a particular purpose. In no event shall Advantage-Devices GmbH be liable for any incidental, special, indirect, or consequential damages whatsoever included but not limited to lost profits arising out of errors or omissions in this manual or the information contained herein.

Advantage-Devices GmbH products are not designed, intended, authorized or warranted for use as components in systems intended for surgical implant into the body, or in other applications intended to support or sustain life, or in any other application in which the failure of a Advantage-Devices GmbH. product could create a situation where personal injury, death, or severe property or environmental damage may occur. Advantage-Devices GmbH reserves the right to discontinue or make changes to its products at any time without notice.

Advantage-Devices and the Advantage-Devices logo, and combinations thereof are registered trademarks of Advantage-Devices GmbH DSTni is a trademark of Lantronix, Inc. All other product names, company names, logos or other designations mentioned herein are trademarks of their respective owners.

XPort is a trademark of Lantronix. Ethernet is a trademark of XEROX Corporation. UNIX is a registered trademark of The Open Group. Windows 95, Windows 98, Windows 2000, Windows NT, and Windows XP are trademarks of Microsoft Corp. Netscape is a trademark of Netscape Communications Corporation.

Advantage-Devices

Otto-Röhm-Strasse 69
64293 Darmstadt, Germany
Phone: +49 6151 6294871

Technical Support

Phone: +49 6151 6294871
Fax: +49 6151 817329
On-line: www.advantage-devices.com

Disclaimer and Revisions

The information in this guide may change without notice. The manufacturer assumes no responsibility for any errors that may appear in this guide.

Date	Rev.	Author	Comments
11/23/04	A	GR	Preliminary Release
06/02/05	B	GR	Notes for Device Installer and Product Code
01/03/06	C	GR	Notes for XPort-MB and NET232-MB

Warranty

Advantage-Devices warrants each product to be free from defects in material and workmanship for a period of ONE YEAR after the date of shipment. During this period, if a customer is unable to resolve a product problem with Advantage-Devices Technical Support, a Return Material Authorization (RMA) will be issued. Following receipt of a RMA number, the customer shall return the product to Advantage-Devices, freight prepaid. Upon verification of warranty, Advantage-Devices will -- at its option -- repair or replace the product and return it to the customer freight prepaid. If the product is not under warranty, the customer may have Advantage-Devices repair the unit on a fee basis or return it. No services are handled at the customer's site under this warranty. This warranty is voided if the customer uses the product in an unauthorized or improper way, or in an environment for which it was not designed.

Advantage-Devices warrants the media containing software and technical information to be free from defects and warrants that the software will operate substantially for a period of 60 DAYS after the date of shipment.

In no event will Advantage-Devices be responsible to the user in contract, in tort (including negligence), strict liability or otherwise for any special, indirect, incidental or consequential damage or loss of equipment, plant or power system, cost of capital, loss of profits or revenues, cost of replacement power, additional expenses in the use of existing software, hardware, equipment or facilities, or claims against the user by its employees or customers resulting from the use of the information, recommendations, descriptions and safety notations supplied by Advantage-Devices. Advantage-Devices liability is limited (at its election) to:

1) refund of buyer's purchase price for such affected products (without interest)

2) repair or replacement of such products, provided that the buyer follows the above procedures.

There are no understandings, agreements, representations or warranties, expressed or implied, including warranties of merchantability or fitness for a particular purpose, other than those specifically set out above or by any existing contract between the parties. The contents of this document shall not become part of or modify any prior or existing agreement, commitment or relationship.

Table of Contents

1. Introduction 1-1
1.1 Modbus 1-1
1.2 Extended Modbus System Example 1-2
1.2.1 Modbus/TCP Master Talking to Modbus/TCP Slave 1-2
1.2.2 Modbus/TCP Master Talking to Modbus/RTU Serial Slave 1-3
1.2.3 Modbus/RTU Serial Master Talking to Modbus/TCP Slave 1-3
1.2.4 Modbus/RTU Serial Master Talking to Modbus/RTU Serial Slave 1-3

2. Configuring Modbus 2-1
2.1 Network Protocols 2-1
2.2 Packing Algorithm 2-1
2.3 IP Address 2-1
2.4 Configuration Methods 2-1
2.5 Device Server's IP Address 2-2
2.6 Device Installer 2-2
2.6.1 RUN Device Installer 2-3
2.6.2 Device Found 2-3
2.7 Using the Setup Mode Screen 2-4
2.8 Basic Commands (D/S/Q) 2-5
2.8.1 Default Settings (D) 2-5
2.8.2 Save (S) 2-5
2.8.3 Quit Without Saving (Q) 2-5
2.9 Network/IP Settings 2-5
2.9.1 IP Address 2-6
2.9.2 Set Gateway IP Address (Y/N) 2-6
2.9.3 Set Netmask (N for default) 2-6
2.9.4 Telnet Configuration Password 2-6
2.10 Serial and Mode Settings 2-6
2.10.1 Attached Device (1=Slave, 2=Master) 2-6
2.10.2 Serial Protocol (1=Modbus/RTU, 2=Modbus/ASCII) 2-6
2.10.3 Interface Type (1=RS232 2=RS422/RS485+4-wire 3=RS485+2-wire) 2-6
2.10.4 Enter Serial Parameters (9600,8,N,1) 2-7
2.11 Modem/Configurable Pin Settings 2-7
2.11.1 CP1 Function 2-7
2.11.2 CP2 Function 2-8
2.11.3 CP3 Function 2-8
2.12 Advanced Modbus Protocol Settings 2-8
2.12.1 Slave Address (0 for auto, or 1..255 fixed otherwise) 2-8
2.12.2 Allow Modbus Broadcasts (1=Yes 2=No) 2-9
2.12.3 Use MB/TCP 00BH/00AH Exception Responses (1=No 2=Yes) 2-9
2.12.4 Disable Modbus/TCP pipeline (1=No 2=Yes) 2-9
2.12.5 Character Timeout (0 for auto, or 10-6950 msec) (50) 2-10
2.12.6 Message Timeout (200-65000 msec) (5000) 2-10

2.12.7 Serial TX delay after RX (0-1275 msec) (0) 2-10
2.12.8 Swap 4x/0H to get 3x/1x (N) 2-10
2.13 Unit ID to IP Address Lookup Table 2-11
2.13.1 Close Idle TCP sockets after (3-60 sec, 0=leave open) 2-11
2.13.2 Redundant Entry Retries after (15-60 sec. 0=disable feature) 2-12
2.13.3 A)dd, D)elete, E)xit Select Function 2-12
2.13.4 Modbus Address From/To 2-12
2.13.5 Slave IP Address 2-12

3. Monitor Mode and Firmware Upgrade 3-1

4. Troubleshooting 4-1
4.1 Troubleshooting Software 4-1
4.2 How fast can I poll? 4-2
4.3 I cannot get a slave response 4-3
4.4 Only Slave ID #1 can be polled 4-3
4.5 Every 2nd poll seems to fail 4-3

5. Technical Support 5-1

1. Introduction

This manual provides detailed operation and setup information for the XPort-03 Device Server with Modbus firmware, known as XPort-MB. This manual also refers to the XPort-MB as a Device Server.

The default protocol found in an XPort-03 device server is Standard Tunneling protocol, a serial protocol used to connect intelligent devices to the Ethernet. Changing the Standard Tunneling protocol to Modbus will change the setup configuration menus and dialogs. Therefore, this manual provides Modbus protocol specific information for the XPort-MB device server and the NET232 MB Adapter.

The NET232 Adapter, designed by Advantage-Devices, uses the XPort-03 Device Server. The NET232 is a complete RS232 Serial to Ethernet interface. The Modbus version of this product is the NET232 MB.

1.1 Modbus

When it comes to planning data communication for open, multi-vendor industrial control systems, Modbus[1] is the undisputed first choice of end users and integrators alike. The Modbus/RTU protocol defines how a "master" device polls one or more "slave" devices to read and write data in real time over RS-232, RS-422, or RS-485 serial data communication. Although not the most powerful protocol available, its rare simplicity allows not only rapid implementation but also enough flexibility to be applied in a large number of industrial situations. Modbus/TCP, an extension of Modbus/RTU, defines how Modbus/RTU and Modbus/ASCII messages are encoded within and transported over TCP/IP-based networks. Modbus/TCP is just as simple to implement and flexible to apply as the original Modbus/RTU. You can find the specification for both online at www.modicon.com.

The XPort-MB Device Server allows users to integrate new and existing Modbus/RTU and Modbus/ASCII serial devices to newer TCP/IP network-based devices. The next section describes a system that integrates four Modbus/RTU devices with four Modbus/TCP devices.

[1] Modbus is a registered trademark of Schneider Automation.

1.2 Extended Modbus System Example

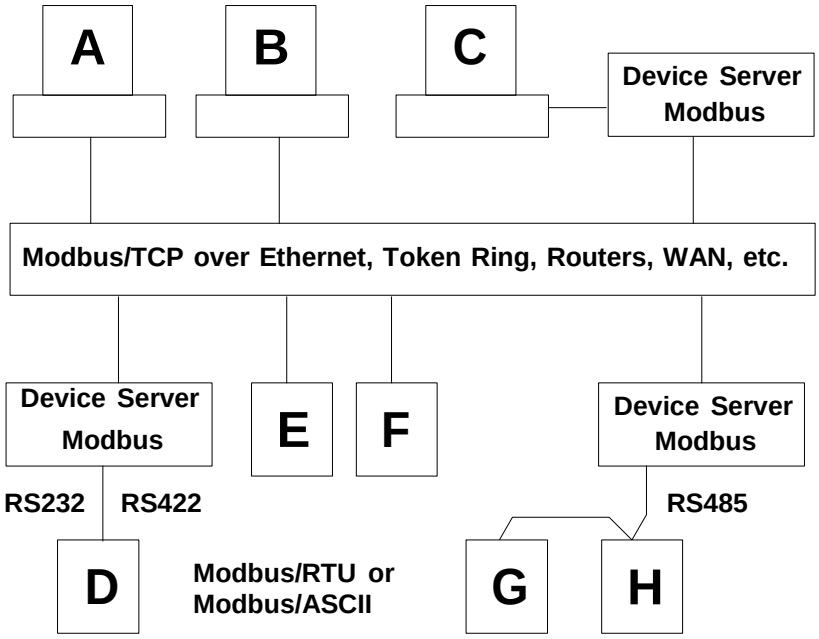


Figure 1 - Extended Modbus System Example

In Figure 1, we can see four specific styles of Modbus operations. Modbus/RTU devices are traditionally split into two groups.

Modbus slave devices generally are the workhorse devices. Often industrially hardened, they tirelessly perform their tasks 24 hours a day, 365 days a year. They perform tasks such as flow metering, temperature control, batch loading, or even running entire automated assembly lines. The slave devices are not called "slaves" because they work all the time; they are called slaves because as far as the data communications is concerned, they function as passive servers. Modbus slave devices passively sit and wait for a remote Modbus master device to ask them to report existing data values (Read) or accept new data values (Write).

Modbus master devices generally are higher-level computers, devices in which data and software are very important. The most common examples of Modbus master devices are the "Human-Machine-Interface" (HMI) computers, which allow human operators to monitor, adjust, and maintain the operations of the field devices. Modbus master devices are clients that actively go out and "Read" from and/or "Write" to remote Modbus slave devices to monitor or adjust slave behavior.

1.2.1 Modbus/TCP Master Talking to Modbus/TCP Slave

Devices A, B, E, and F are all new Modbus/TCP devices, which are improved over Modbus/RTU (see more about Modbus/RTU limitations below). All 4 devices can function concurrently as both Modbus master and Modbus slave. Both computers A and B can treat controller E as a slave, polling data in real-time. Yet controller E can also act as a master and poll data from controller F, which can in turn also act as a master to write alarm data directly up to computers A and B to alert the operators to the alarm condition. Traditional Modbus/RTU requires slave devices even with life threatening alarm conditions to sit patiently and wait for a remote master to poll the specific data that caused the alarm condition.

It is really revolutionary for such a simple and flexible protocol as Modbus to offer such functionality. Therefore, Modbus/TCP offers exciting new design options for industrial users, which the Device Servers extend to traditional Modbus/RTU serial devices.

1.2.2 Modbus/TCP Master Talking to Modbus/RTU Serial Slave

Devices D, G, and H are traditional Modbus/RTU slave devices. Device D uses a point-to-point electrical interface like RS-232. This allows only a single Modbus/RTU master to talk to device D. However, the Device Server makes device D appear on the Modbus/TCP network as a full Modbus/TCP slave device. All Modbus/TCP enabled devices, A, B, E, and F, can actively share access to slave device D. A limitation in traditional Modbus/RTU implementation expects devices to be dedicated as either master or slave devices, so device D can only act as a Modbus slave.

Devices G and H are different from device D. They share a single RS-485 "multi-drop" line that strictly limits them to act as slaves to a single Modbus/RTU master. However, a little of the new Modbus/TCP and Device Server magic still applies↓all Modbus/TCP enabled devices A, B, E, and F can actively share access to both slave devices G and H. The Device Server manages and coordinates the shared access. In fact, the Device Server allows up to eight concurrent Modbus masters to share access to the slaves.

1.2.3 Modbus/RTU Serial Master Talking to Modbus/TCP Slave

Device C is a traditional Modbus/RTU master device. Yet the Device Server makes device C appear to the TCP/IP network as a Modbus/TCP master↓plus all of the Modbus/TCP slaves on the TCP/IP network (A, B, D, E, F, G, and H) appear as traditional Modbus/RTU slave devices. The only limitation is the traditional Modbus/RTU assumption that device C is dedicated as a master only. Therefore Modbus/TCP master devices A, B, E, and F cannot treat device C as a Modbus/TCP slave.

1.2.4 Modbus/RTU Serial Master Talking to Modbus/RTU Serial Slave

Finally, master device C can poll traditional Modbus/RTU slave devices D, G, and H as if they were directly multi-dropped on an attached RS-485 line. The Device Server transparently bridges traditional Modbus/RTU devices across any TCP/IP network. This means users can start implementing for Modbus/TCP long before all of their required products exist with Modbus/TCP and network interfaces.

2. Configuring Modbus

2.1 Network Protocols

The XPort-MB Device Server uses TCP/IP protocols for network communication. The supported standards are: ARP, UDP, TCP, ICMP, Telnet, TFTP, DHCP, and SNMP. For transparent connections, TCP/IP (binary stream) or Telnet protocols are used. Firmware upgrades can be made with the TFTP protocol.

The industrial protocol defines addressing, routing and data block handling over the network. The TCP (transmission control protocol) assures that no data is lost or duplicated, and that everything sent into the connection on one side arrives at the target exactly as it was sent.

For typical datagram applications where devices interact with others without maintaining a point-to-point connection, UDP datagram is used.

2.2 Packing Algorithm

Traditional Modbus/RTU requires a “character time-out” to signal the end of a Modbus/RTU packet. This stretches out the overall response cycle. Fortunately, the XPort-MB Device Server uses an intelligent length-predictive algorithm to detect the end of standard Modbus messages. This allows better performance plus the XPort-MB Device Server falls back to using a user definable “character time-out” to manage non-standard or user-defined Modbus functions.

2.3 IP Address

Every device connected to the TCP/IP network including the XPort-MB Device Server must have a unique IP address. When multiple Modbus devices share a single IP, then Modbus/TCP includes an additional address called the Unit ID.

When the XPort-MB Device Server is receiving Modbus/TCP messages from remote masters, the Unit ID is converted to use in the Modbus/RTU message as the slave address.

When the XPort-MB Device Server is receiving Modbus/RTU messages from local serial masters, a user-defined lookup table is used to match the 8-bit Modbus slave address to a remote IP address. The Modbus slave address received is used as the Unit ID.

2.4 Configuration Methods

The XPort-MB Device Server can be configured using remote or local methods. Either use an ASCII terminal or a terminal emulation program to locally access the serial port, or use a Telnet connection to port 9999 to configure the unit over the network.

The XPort-MB Device Server configuration is stored in nonvolatile memory and is retained without power. The configuration can be changed any time. The XPort-MB Device Server performs a reset after the configuration has been changed and stored.

2.5 Device Server's IP Address

The XPort-MB Device Server is shipped with a default IP address of 0.0.0.0, which automatically enables DHCP within the XPort-MB Device Server.

With a DHCP-enabled Device Server, if there is a DHCP server to respond to the Device Server's request when it's booting up, the Device Server will then get an IP address, a gateway address, and a subnet mask from the DHCP server. These addresses will not be shown in the Device Server's Setup (configuration) screens (you will still see 0.0.0.0), however if you enter the Monitor Mode and from 0> prompt, type NC (upper case), the IP configuration of the Device Server will display. (See the Monitor Mode and Firmware Upgrade chapter.)

2.6 Device Installer

The XPort-MB is shipped without software or hardware manuals. Device Installer software and documentation can be downloaded from the Lantronix web site. See www.lantronix.com. The XPort-MB user manual can be downloaded from the Advantage-Devices web page.

If you purchased a NET232-MB, Device Installer software and user manuals are provided on the supplied CD. See the *NET232 User Manual* for Device Installer installation instructions. See the *Device Installer User Manual* for operation details. Since the NET232 MB has different firmware, the operation of Device Installer will be different.

Device Installer uses .NET Framework to adapt XPort embedded servers for Web services. Your system must have .NET Framework installed for Device Installer to work properly. XPort-MB users can download .NET Framework from the Lantronix web page. See www.lantronix.com. NET232-MB users can install the software from the supplied CD.

Device Installer looks for devices on the network and identifies them according to a device code in the firmware. Device Installer looks at the device code from all devices found and compares them to a list of acceptable codes. The list is the file ProductInfoBase.txt. The list from the Lantronix web page may not include the XPort-MB. You will have to enter a new device code for the XPort-MB.

The device codes are stored in a "ProductInfoBase.txt" file under Program Files, Lantronix, DeviceInstaller. Open the text file in Notepad and add the following line:

```
"XA","XPort-03 Modbus"
```

Do not enter any extra spaces or lines. Save the file and start Device Installer. The program will locate the XPort-MB and display ***XPort-03 Modbus***.

2.6.1 RUN Device Installer

Click the Start button on the Task Bar and select **Programs\Lantronix\Device Installer**. Select **Device Installer** from the option list. The Device Installer main dialog box appears.

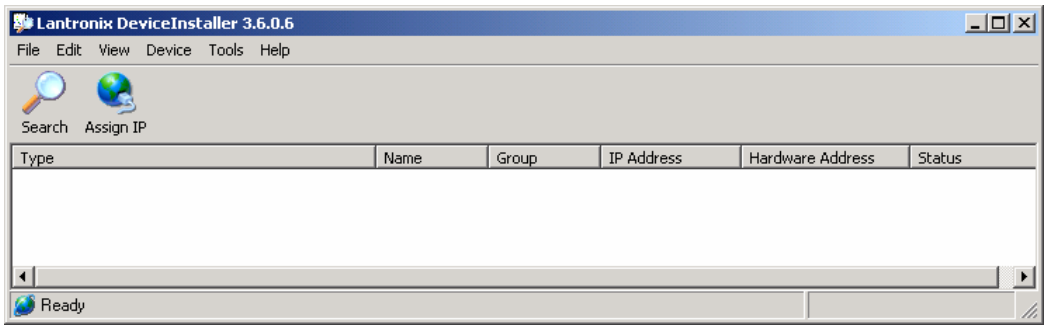


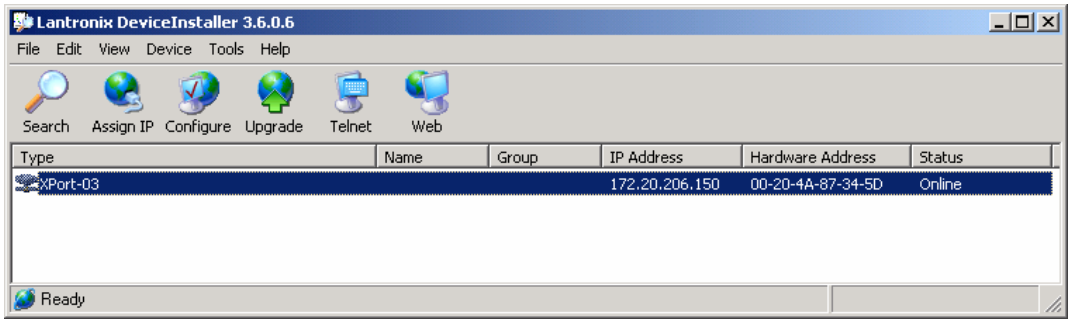
Figure 2 - Device Installer Dialog Box

To search for devices, click the **Search** icon or select **Search F5** from the Device menu.

2.6.2 Device Found

The following dialog box shows a device found on the network, with the IP addresses assigned by the DHCP server. The device IP Address is normally set to 0.0.0.0 at the factory. The Hardware Address is an individual permanent address assigned to a particular device on the network. The Hardware Address can be found on the product label.

Note: The device found in the dialog box is a standard XPort-03 without Modbus firmware.



You can use the **Assign IP, Upgrade** and the **Telnet** buttons.

Note: The Web manager page has been deleted from the firmware so the Web button does not work.

Note: Do NOT use the Configure button for changing the Baud Rate.

2.7 Using the Setup Mode Screen

1. From the DeviceInstaller Configuration Utility, click the Telnet button to open a Telnet connection to the XPort-MB Device Server. You'll see the following lines, which tell you the Device Server's Ethernet hardware address (or HW MAC).

```
*Modbus/TCP to RTU Bridge
MAC address  00204A81DA6D
Software version 02.2b1 (040728) XPTEX
Press Enter to go into Setup Mode, wait to close
```

2. Within 5 seconds, press Enter to display the Setup (configuration) Mode screen. Here you can change the parameters that define how the XPort-MB Device Server does its job.

Note: When you set up a new unit, and especially if you just reflashed the unit with a new firmware type, we recommend that you reset all of the parameters to the factory defaults.

3. To reset the parameters to the factory defaults, type **D** on the command line and press Enter. The default parameters display.
4. Select an option on the menu (1-4) by typing the number of the option.

Note: Selecting the serial protocol to be Modbus Master will add an additional item to the list.

5. To enter a value for a parameter, type the value and press Enter, or to confirm a default value, press Enter.
6. Review your entries.
7. You have the following options:

   To save the configuration and exit, type **S** on the command line and press **Enter**. This saves the parameters to EEPROM.

Caution: ***DO NOT POWER CYCLE*** *the unit too fast after doing this. Allow the unit to reboot naturally one time first.*

To quit without saving, type **Q** on the command line and press Enter. The unit reboots.
To restore the default values, type **D** on the command line and press Enter.

```
Model: Device Server Plus+! (Firmware Code:XA)

Modbus/TCP to RTU Bridge Setup
1) Network/IP Settings:
     IP Address ..................... 192.168.100.77
     Default Gateway  . . . . . --- not set ---
     Netmask  . . . . . --- not set ---
2) Serial & Mode Settings:
     Protocol . . . . . Modbus/RTU,Slave(s) attached
     Serial Interface . . . . . 9600,8,N,1,RS232
3) Modem/Configurable Pin Settings:
     CP1 . . . . Status LED Output
     CP2 . . . . DTR Output Active with connection
     CP3 . . . . Diagnostic LED Output
4) Advanced Modbus Protocol settings:
     Slave Addr/Unit Id Source  . . Modbus/TCP header
     Modbus Serial Broadcasts . . . Disabled (Id=0 auto-mapped to 1)
     Modbus/TCP pipeline  . . . . . Enabled (new MB/TCP requests queued in FIFO)
     MB/TCP Exception Codes . . . . Yes (return 0x0A and 0x0B)
     Char, Message Timeout  . . . . 00050msec, 05000msec

D)efault settings, S)ave, Q)uit without save
Select Command or parameter set (1. . . 4) to change:
```

Figure 3 - Setup (Configuration) Mode Screen

2.8 Basic Commands (D/S/Q)

The main Device Server configuration menu is shown above. The Device Server offers three basic options.

2.8.1 Default Settings (D)

Entering **D** resets all parameters to the factory default as shown above. Only the IP address is not changed. Although not required, selecting this option immediately after reloading the firmware and saving it ensures that the unit is reset.

2.8.2 Save (S)

Entering **S** saves the currently displayed parameter settings into non-volatile memory and exits configuration mode. This option will trigger a reset.

2.8.3 Quit Without Saving (Q)

Entering **Q** throws away any parameter changes you have made and exits configuration mode. This option will trigger a reset.

2.9 Network/IP Settings

Select **1** to configure the Device Server's network parameters.

The following values can be set/changed. To understand and select the appropriate values, consult one of the many TCP/IP books available today and your network administrator.

2.9.1 IP Address

The IP address must be set to a unique value on your network. If you are not familiar with IP addressing on your network, please consult your system administrator. Please refer to the User Guide for your Device Server for more details about IP addresses.

If the Device Server is set to an address already in use, it will display an error code with the LEDs and it will not operate properly. If you understand and plan to use DHCP, set the IP to 0.0.0.0 to activate DHCP.

2.9.2 Set Gateway IP Address (Y/N)

Most users could select **N** for this case. You only need to choose **Y** if your Device Server must communicate to remote TCP/IP networks through a router or gateway. If you select **Y**, you must also enter the IP address of the default gateway within your local network.

2.9.3 Set Netmask (N for default)

Most users could select **N**, which causes the Device Server to automatically use the standard netmask appropriate for the IP address you've entered. Users who want a non-standard netmask need to enter the new subnet mask in the traditional form, for example, 255.255.248.000. The selecting of correct IP ranges and subnet masks IS a large enough topic to fill a whole book – we cannot cover it here.

2.9.4 Telnet Configuration Password

The telnet configuration password can be set to disable unauthorized access to the setup menu via a Telnet connection to port 9999. To access the setup menu through the serial port, it is not necessary to enter the password.

2.10 Serial and Mode Settings

Select **2** to change the basic serial parameters. The following values can be set/changed.

2.10.1 Attached Device (1=Slave, 2=Master)

As mentioned in the introduction, Modbus/RTU devices are defined as either slave or master devices. Type **1** if the attached device is a slave (such as controller or PLC) or **2** if the attached device is a master (such as a computer running graphical human-machine-interface (HMI) software).

2.10.2 Serial Protocol (1=Modbus/RTU, 2=Modbus/ASCII)

Serial Modbus comes in two forms. Modbus/RTU uses 8-bit data bytes to send binary information. However, some devices cannot handle 8-bit data bytes, so Modbus/ASCII is used. Modbus/ASCII is a slower protocol where each 8-bit data byte is converted to 2 ASCII characters. Since the Device Server converts both to and from Modbus/TCP fully, you can mix any combination of RTU and ASCII devices on a Modbus/TCP network. So a Modbus/RTU Master attached to one Device Server can remotely access a Modbus/ASCII slave attached to another Device Server.

2.10.3 Interface Type (1=RS232 2=RS422/RS485+4-wire 3=RS485+2-wire)

This allows the Device Server to deal with the software-related details of using RS-232, RS-422, and RS-485.

Note: The NET232/USB MB device is currently available with only RS232 interface.

2.10.4 Enter Serial Parameters (9600,8,N,1)

Enter the baud rate, data bits (7/8), parity (N/O/E), and stop bits (1/2) in the classic "DOS Mode Command" style. Examples are: 9600,8,E,1 or 1200,7,O,2. This setting must match the setting on the attached Modbus device.

You will be warned if you try to set an unsupported combination of settings.

Note: After reset, the device server will check the serial port at 9600, 8, N, 1 for 5 seconds.

Note: Do NOT use the Configure button for changing the Baud Rate.

2.11 Modem/Configurable Pin Settings

The Modbus Master/Slave functionality on the XPort-MB is very similar to the Modbus implementation on other device servers. The major difference is that the configurable pins on the XPort-MB (CP1-3) can be configured from the setup menu.

2.11.1 CP1 Function

This configurable pin can be set to:

```
CP1 Function (1=Unused, 2=Status LED Output, 3=RTS Output)
```

Note: Status LED Output is not valid for NET232-MB devices.

The **Status LED Output** will provide a status indicator. Normally, this signal is tied to an LED on your device. The LED will blink 4 times, then be off for two seconds to indicate the setup mode is active through the serial port.

Selecting **RTS Output** for CP1 will prompt the user for additional RTS and flow control options.

```
RTS Mode 1 = Fixed, 2 = Active with transmit.
```

Selecting **Fixed** will complete the CP1 setup and the menu drops to CP2.

Selecting **Active with Transmit** will display the following option:

```
Delay after output RTS (0-1275 msec, 5ms resolution) (0)
```

After the NET232/USB MB asserts the RTS/CTS signal, it delays from 0 to 1275 msec before continuing. Normally this is set to **0**. Only set a value here if your device, modem or cable is non-standard.

```
Wait for CTS (CP3) to go active (N)
```

Selecting **Y** to **Wait for CTS** will display the following option:

```
Delay after CTS going active (0-1275 msec, 5ms resolution)  (0)
```

After the NET232/USB MB sees the modem assert an RTS/CTS response input, it delays from 0 to 1275 msec before transmitting. If the unit waits without seeing a valid response from the modem, it will return the Modbus exception response 0x0B (hex) to the Modbus/TCP requesting master.

```
Delay dropping RTS after TX (0-1275 msec, 5ms resolution)
```

After the unit completes transmission, it delays from 0 to 1275 msec before dropping the RTS/CTS output.

Selecting **Wait for CTS** will cause CP3 to be auto-configured for CTS Input. You will not be able to configure CP3 and the following message will appear:

```
CP3 Function already configured for CTS Input
```

2.11.2 CP2 Function

This configurable pin can be set to:

```
CP2 Function (1=Unused, 2=DTR Output, 3=RS485 Output Enable.
```

Note: CP2 must be set to Unused for NET232-MB products.

Selecting **DTR Output** for CP2 will prompt you for additional DTR control options.

```
DTR Mode (1=Fixed, 2=Active with connection)
```

Selecting **RS485 Output Enable** for CP2 will prompt you for additional control options.

```
Invert RS485 Output Enable(active low) (N)
```

The RS485 Output Enable function is used for controlling an external RS485 line driver when in RS485 2-wire mode and this output can be configured for active high (default) or active low.

2.11.3 CP3 Function

This configurable pin can be set to:

```
CP3 Function (1=Unused, 2=Diagnostic LED Output)
```

CP3 can be forced to CTS Input by answering '**Y**' to the **Wait for CTS** option under the CP1 Function menu for **RTS Output**.

Note: Selecting Diagnostic LED Output is not a valid option for NET232-MB products.

The Diagnostic LED Output is a status indicator to indicate the unit is in setup mode. When used in conjunction with the Status LED Output, the following conditions can be monitored.

Condition	LED 3	LED 1
No errors	OFF	ON
Network controller error	ON	Blink 3x/4 sec OFF
Duplicate IP address present	ON	Blink 5x/4 sec OFF
No DHCP response	Blink 2x/sec	Blink 5x/4 sec OFF
Setup menu active	Blink 2x/sec	See Note.

Note: During a Telnet connection, CP1 LED (Status LED) is ON. For a serial port connection, CP1 LED (Status LED) blinks for 2 seconds, then OFF for 2 seconds. (It appears as 4 blinks, then OFF for 2 seconds)

2.12 Advanced Modbus Protocol Settings

Changing these parameters takes a bit of thought and planning.

2.12.1 Slave Address (0 for auto, or 1..255 fixed otherwise)

Modbus/TCP includes a Unit ID field, which is used to address multiple Modbus slaves at a single IP address. Unfortunately, some first generation software drivers assumed a single slave at each IP and always set the Unit ID field to 0. This causes the Device Server problems because it requires the Unit ID for the Modbus/RTU "Slave Address". To support these older applications, the Device Server allows you to force a fixed address for Modbus/RTU and Modbus/ASCII, but note that this restricts you to a single serial slave device per Device Server.

Setting this value to 0 causes the Device Server to use the Modbus/TCP Unit ID as received. Setting it to any other address causes the Device Server to always use the set value as a fixed address.

2.12.2 Allow Modbus Broadcasts (1=Yes 2=No)

This actually relates to the previous issue. The default is 2/No, in which case Device Server always assumes a Modbus/TCP "Unit ID" of 0 really means Modbus slave address 1. Answering No here is like setting a fixed address of 1 (parameter above), except the fixed address is only used if the Modbus/TCP "Unit ID" is 0.

Note: In the current software version for Device Server, a true Modbus broadcast is only supported when a serial slave device is attached. A Modbus broadcast from a serial master device is discarded regardless of this parameter setting.

2.12.3 Use MB/TCP 00BH/00AH Exception Responses (1=No 2=Yes)

Traditional serial Modbus uses silence to signal some errors. While this works well with direct serial lines, it causes serious problems on a TCP/IP wide-area-network where delays are not so predictable. See the Troubleshooting chapter for a full discussion.

Setting this to **1/No** causes the Modbus bridge to behave like a traditional Modbus serial slave – it answers timeouts, unconfigured slave addresses, and CRC errors with silence.

Setting this to **2/Yes** causes the Modbus bridge to return 1 of 2 new exception codes defined in Modbus/TCP.

Consider exception hex 0A (PATH UNAVAILABLE) a "hard" error where a retry is not likely to succeed. It is returned:
If slave-attached – currently never. However, future firmware may allow the user to define the range of valid slave addresses.
If master-attached – if a Modbus request has a slave address that is not configured in the Unit ID to IP mapping table.
If master-attached – if the TCP socket failed to open. This is really a soft-hard error, as the reason the TCP socket failed to open may be transient or a hard configuration error.

Exception hex 0B (TARGET DEVICE FAILED TO RESPOND) should be considered a "soft" error where a retry may succeed. It is returned:
If slave-attached – if the slave didn't answer or the answer contained a CRC error
If master-attached – if a TCP socket is open, but no response was received in the defined message timeout.
If master-attached – if a TCP socket is open, but the remote Modbus/TCP slave/server returned exception 0x0B.

2.12.4 Disable Modbus/TCP pipeline (1=No 2=Yes)

While the Modbus/TCP standard specification requires Modbus/TCP masters/clients to only issue 1 poll at a time, the full-duplex flow-controlled nature of TCP/IP allows them to issue more than one at a time, and the TCP socket will happily buffer them. The Modbus Bridge will fetch them one at a time and answer each in turn. See the Troubleshooting chapter for a full discussion of the problem this can cause.

Setting this to **1/No** causes the Modbus Bridge to allow this queuing or pipeline behavior. This is the safest default setting – only change this to **disable** if you are having problems.

Setting this to **2/Yes** causes the Modbus Bridge to always fetch the newest request from the TCP buffer – all older requests are discarded. This allows a Modbus/TCP master/client to issue new requests without risking building up a stale queue of waiting requests.

2.12.5 Character Timeout (0 for auto, or 10-6950 msec) (50)

This sets the timeout between characters received. Official Modbus/RTU defines a 3.5 character time-out, but complex devices have various interrupts that can cause 5 to 10 character "pauses" during transmission. A safe value for general use with Modbus is 50 msec. A setting of 0 will force the Device Server to automatically calculate a minimum timeout based on the baud rate.

Note: Setting this value lower than 50 msec will not improve performance and may even make performance worse. The Device Server uses an intelligent length-predicting algorithm to detect end-of-message in Modbus/RTU. This "character timeout" is only used with user-defined or non-standard Modbus functions.

2.12.6 Message Timeout (200-65000 msec) (5000)

This sets the timeout for a response from a connected slave both serially and by TCP/IP.

2.12.7 Serial TX delay after RX (0-1275 msec) (0)

This feature inserts a delay between the Modbus/TCP master requests. The first request is sent out of the serial port of the Device Server to the Modbus slave. When the slave's response enters the serial port of the Device Server, it triggers this timer. After the specified delay is reached, the next master request is allowed to pass through the serial port of the Device Server, and the timer is reset. This feature is particularly useful when using RS485 2-wired serial protocol. The delay gives ample time for the RS485 slave devices to turn their transmitters off and their receivers back on. Normally this should be set to 0 – change it only if you are having problems.

2.12.8 Swap 4x/0H to get 3x/1x (N)

This setting will convert holding registor (4x) data to input registor (3x) data. It also converts coil (0x) and contact data (1x). This feature is useful for Modicon I/O scanners.

2.13 Unit ID to IP Address Lookup Table

Selecting the Serial Protocol to be Modbus Master will add an additional item to the configuration option list. The new option is the Unit ID to IP Address Table and appears as option 5.

The new menu appears like this:

```
1) Network/IP Settings:
     IP Address ..................... 192.168.100.77
     Default Gateway  . . . . . --- not set ---
     Netmask  . . . . . --- not set ---
2) Serial & Mode Settings:
     Protocol . . . . . Modbus/RTU,Master(s) attached
     Serial Interface . . . . . 9600,8,N,1,RS232
3) Modem/Configurable Pin Settings:
     CP1. . . . .Not Used
     CP2. . . . .Not Used
     CP3. . . . .Not Used
4) Advanced Modbus Protocol settings:
     MB/TCP Exception Codes . . . . Yes (return 00AH and 00BH)
     Char, Message Timeout  . . . . 00050msec, 05000msec
5) Unit ID -> IP Address Table
     Close Idle Sockets . . . . . 10sec
     Redundant Entry Retry  . . . . Feature Disabled

D)efault settings, S)ave, Q)uit without save
Select Command or parameter set (1. . . 5) to change:
```

Since serial Modbus uses 8-bit slave addresses and a TCP/IP network requires 32-bit IP addresses, the Device Server uses this table to map an 8-bit address into an IP/Unit ID combination. The 8-bit address is used to both select the desired IP and as the Unit ID sent. The table holds 8 entries, and any Modbus slave address not found in the table returns an exception response to the master (if enabled).

Below is an example of adding an entry. Select **5** to edit/view settings.

```
Close Idle TCP sockets after (3-60 sec, 0=leave open) (10)
Redundant entry retries after (15-60 sec. 0=disable feature) (0)
(Set 4th octet to 0 to use Slave Address as part of IP)

    1):      001-100: 192.168.000.000+SLV
    2):      101-199: 192.168.000.150

A)dd, D)elete, E)xit - select function  A
        Modbus addr from  (102)
        Modbus addr   to  (102) 255
        Slave IP address (192) 172.(168) 16.(000) 123.(000)

    1):      001-100: 192.168.000.000+SLV
    2):      101-199: 192.168.000.050
    3):      200-255: 172.016.123.000+SLV

A)dd, D)elete, E)xit - select function
```

Figure 4 - Unit ID to Address Lookup Table Example

2.13.1 Close Idle TCP sockets after (3-60 sec, 0=leave open)

Sockets are opened as required. Entering a 0 holds a single socket open to the last remote Modbus/TCP slave accessed.

Otherwise enter values 3 to 60 to automatically close the last socket after 3 to 60 seconds of idle time.

2.13.2 Redundant Entry Retries after (15-60 sec. 0=disable feature)

Enter the time in seconds for redundant entry retries or set to 0 to disable the feature.

2.13.3 A)dd, D)elete, E)xit Select Function

You can either add or delete entries in the IP address table. They are automatically sorted into increasing order. Enter E when you are satisfied with the table to return to the main menu.

2.13.4 Modbus Address From/To

This is the minimum/maximum Modbus slave address (inclusive) to forward to this IP address.

2.13.5 Slave IP Address

This is the IP address of the remote Modbus/TCP slave. Note the two different ways these IP are interpreted. In the configuration example above you'll see the following results:

Polls to Slave #12 will go to IP 192.168.0.12 with Unit ID 12.
Polls to Slave #70 will go to IP 192.168.0.70 with Unit ID 70.
Polls to Slave #112 will go to IP 192.168.0.50 with Unit ID 112.
Polls to Slave #155 will go to IP 192.168.0.50 with Unit ID 155.
Polls to Slave #201 will go to IP 172.16.123.201 with Unit ID 201.
Polls to Slave #244 will go to IP 172.16.123.244 with Unit ID 244.

Setting the last/4th IP octet to zero is interpreted as a signal to use the Slave ID as part of the IP. This allows a Modbus/RTU master to access up to 255 remote Modbus/TCP slaves. Setting the last/4th octet of the IP to 1-254 causes all slave polls in this group to be sent to the same IP. 255 is not accepted as the last/4th IP octet.

3. Monitor Mode and Firmware Upgrade

The easiest way to upgrade your protocol firmware (or "reflash") is to use the DeviceInstaller Utility.

Note: This procedure should be done with caution.

There are important differences between the industrial protocol firmware (Modbus) files and standard firmware files. Although the NET232/USB (XPort) and NET232/USB MB (XPort MB) hardware may be the same, you will not be able to download a standard firmware file to the Modbus version and vice versa. These firmware files are rejected with the error "Sorry, that firmware not supported." The industrial firmware also has blocked the "SF" command within the Monitor.

4. Troubleshooting

Fortunately, using the Modbus Bridge firmware is normally painless and easy to do.

Unfortunately in some situations it won't be easy or painless at all! In those situations you'll find it difficult to troubleshoot without an in-depth knowledge of Modbus and the system dynamics of polling.

Some general guidelines for trouble-shooting:

- Start polling slowly and work your way up.
- If you're using a custom cable, consider instead first **starting** with a simple, home-made adapter that makes your Device Server's RS-232 port look like a 9-pin DTE port such as on a computer. Then you can use a **known-good** cable to connect your device. Starting with a custom cable is 99 percent guaranteed to be a frustration – first prove everything is set up correctly, then use your custom cable as the final test.
- At a minimum you should have access to something to watch the serial line. Most host applications do a rather POOR job of explaining errors to you. We cannot count how many times we've had customers complain of "No Response," only to find out the device actually **did** respond. It was just the host application declaring "No response" instead of the true "response not understood."

4.1 Troubleshooting Software

Specializing in testing and diagnostic tools for Developers, WinTECH Software offers several products designed for the integration and troubleshooting of communications systems. All applications available from their site are fully functional time-limited demos, and may be freely downloaded and distributed for evaluation purposes. Developed by a Windows Developer for professional use, each application comes complete with an unconditional 30-day money-back guarantee. The web site is www.win-tech.com.

Modscan is a Windows application which operates as a modbus master. It allows you to access and change data points in a connected slave device using either the RTU or ASCII Transmission mode. ModScan is ideally suited for quick and easy compliance testing of the modbus protocol and its built-in display of serial traffic allows effective troubleshooting of field connections.

ModSim32 is a very simple but powerful application for simulating data from modbus slave devices. ModSim is an MDI application which allows you to define multiple blocks of data points which may then be accessed from a connected modbus master. Each document opened within ModSim may be configured to represent data from the same or different slave node thereby providing simulation for an entire group of devices. Useful for complience testing of new master designs or as a quick simulation of a process.

4.2 How fast can I poll?

First, remember that you still have the serial link in there and therefore cannot expect to poll any faster than you could by a direct serial link. In fact, since you are adding a number of queuing systems between your application and device, you may even lose a bit of performance. For example, some download tests showed remote download by Modbus/TCP bridged to Modbus/RTU ran about 20 percent slower than direct download by Modbus/RTU.

But above all remember that the serial speed (or baud rate) consumes the largest amount of time (see the table below). Suppose you issue a Modbus poll for 125 registers. This requires a 255-byte response, which at 19.2kbps requires over 133 msec just to physically shift across the wire, while at 300 baud it takes nearly 10 seconds!

Table 1 - Baud Rate

Baud Rate	Byte/Sec	Bit Time (msec)	Byte Time (msec)	256 Byte Time (msec)	(in sec)
300	30	3.333333	33.333333	8533.333333	8.53
600	60	1.666667	16.666667	4266.666667	4.27
1200	120	0.833333	8.333333	2133.333333	2.13
2400	240	0.416667	4.166667	1066.666667	1.07
4800	480	0.208333	2.083333	533.333333	0.53
9600	960	0.104167	1.041667	266.666667	0.27
19200	1920	0.052083	0.520833	133.333333	0.13
38400	3840	0.026042	0.260417	66.666667	0.07
57600	5760	0.017361	0.173611	44.444444	0.04
115200	11520	0.008681	0.086806	22.222222	0.02

The overall time it takes to poll is the combined sum of these delays:

1. Delay for Master /Client to recognize need for poll
2. Delay to issue and get the poll onto the Ethernet
3. Delay for the poll to cross Ethernet and arrive error-free at the Modbus Bridge device (may include retries and contention)
4. Delay for Modbus Bridge to process and queue Modbus/RTU poll
5. Delay for the serial link to be free (remember other Masters/Clients may be actively polling)
6. Physical delay to shift poll bit-by-bit across the serial link
7. Delay in the device to recognize, process, and start reply
8. Physical delay to shift response bit-by-bit across the serial link
9. Delay for Modbus Bridge to process and queue Modbus/TCP Response
10. Delay for the response to cross Ethernet and arrive error-free at the Master/Client (may include retries and contention)
11. Delay for Master /Client to recognize need for poll

Delays **a** and **k** are defined by your OPC or DDE driver. For example, a driver that runs only once each 55 msec (using the old DOS timer slice) can have a variable delay here of between 0 to 110 msec.

Delays **c** and **i** are defined by the complexity and load of your TCP/IP network. For example, if you're going thru radio or satellite links, these delays routinely amount to 1000 msec (1 sec) or more per poll and another 1000 msec for a response.

Delays **f** and **h** are defined by the baud rate. Assuming an 8 bytes poll and 255-byte response, at 9600 baud this is at least 275 msec, while at 1200 baud this is at least 2200 msec (2.2 sec).

Delay **g** is defined by the device. Oddly enough, the simpler the device, the faster it tends to reply. Some controllers only allocate fixed time slices to process a response from shared memory – for example once each 100 msec.

Delays **d**, **e,** and **i** are defined by the load on the Modbus Bridge. If other Master/Clients are polling, the queuing delay for **e** can be large (the sum of delays **f**, **g,** and **h)** for each earlier poll waiting.

4.3 I cannot get a slave response

Besides the obvious wrong baud rate, there are many possible causes of this:

- Is your cable set up correctly for RS-232?
- An external Signal Ground connection is often required between devices.
- The Modbus Bridge firmware only expects Modbus/TCP from the network. Some applications just pack Modbus/RTU raw in TCP – this is not supported.

4.4 Only Slave ID #1 can be polled

Your application is setting the Modbus/TCP Unit ID field to 0. This causes the Modbus Bridge firmware to automatically map this to 1.

4.5 Every 2nd poll seems to fail

Your device probably cannot accept a new poll as fast as the Modbus Bridge firmware is sending it. Remember, TCP/IP is a full-duplex channel, plus since you can have up to 8 active sockets it is very easy to have a new request already waiting as your last response is being returned. The only solution to this is to slow down your Modbus/TCP masters so they never poll before the last response has been seen. This manually creates the time delay between polls your device expects.

- My Bridge runs fine - for about 10 minutes and then my applications start reporting slaves going off-line.
- My Bridge runs fine – until a slave goes off-line; then I tend to lose all the slaves or they all poll only intermittently.
- Sometimes my Bridge returns the wrong data from the wrong slave.
- After a while, the Bridge seems to take longer and longer to answer – after a few hours, it takes 10 minutes or more for systems changes to propagate up to the Master/Client.

All these relate to the same issue – a mismatch in queuing behavior and expectation by the Master/Client to the new realities of Ethernet. No, it's not the Modbus Bridge behaving poorly. Yes, resetting the Bridge does "fix" the problem (flushes the bloated TCP queues full of stale requests).

The core problem is that the Master/Client is using the old RS-485 serial assumption that no-answer means poll was lost. However, in the Modbus Bridge case, it could also mean the Bridge has not had time to answer (is being over-worked). Also remember that TCP is reliable – the Bridge receives all polls sent without error. The result is that the Master/Client retries, which like throwing gasoline on a fire, makes it harder for the Bridge to catch up.

Here is the scenario that is hurting you:

1. Master sends out MB/TCP Poll #A with a timeout of 1000 msec.

2. Bridge receives the poll, but the serial link is busy so it waits - possibly another MB/TCP master is being serviced or time-outs waiting on off-line stations are creating a backlog of new requests.

3. After approximately 850 msec, the serial link is now free and the Bridge forwards the MB/RTU request.

4. The Bridge receives the response, and since the timeout on the Bridge and Master are not inherently synchronized, the Bridge sends the MB/TCP response into the TCP socket.

5. In the best of times, it may take 5-10msec for this response to actually go down the Bridge's TCP stack, across the wire, and up the master's TCP stack. If a WAN or satellite is involved, it could take 750 msec or longer.

6. Meanwhile, before the Master receives the Response #A, it gives up and makes the Modbus/RTU assumption that the request must have been lost. The Master sends out a new MB/TCP Poll #B.

7. A few msec later, there is a response that looks like a good Response #B, but really is Response #A. If the Master does not use a sequence number (unfortunately many do not) and has forgotten about pending poll #A, it wrongly assumes this is response #B (possibly with catastrophic results if Poll #B was the same size but different register range). **So here is the source of your "Bridge returns the wrong data for wrong slave" problem.**

8. The Master is idle and has no out-standing polls. Yet the Bridge has received Poll #B by reliable TCP/IP. It sends this out to Modbus/RTU slave and gets an answer. The Bridge is doing its job!

9. The Bridge returns Response #B to the master (if the socket is still open) and there it sits in its TCP/IP buffer. The Master is not expecting more responses, so it neither receives nor purges the "extra" response.

10. Master sends Poll #C and magically finds "a response" waiting as soon as it looks in the receive buffer - yet this is stale Response #B received before poll #C was even issued. If the Master does **not** implement Modbus/TCP sequence numbers, then it accepts the response #B as satisfying poll #C. Imagine if the Master is putting out 300 polls per minute (5 polls per second), but the Bridge can only process on average 290 of those per minute and some carry over. After 10 minutes, you may have up to 100 "stale" responses waiting in your Master's TCP buffer. This makes it appear as though there is now a 20 second "lag" in data reaching the Master. **So here is the source of your "data taking longer and longer to propagate to Master/Client" problem.**

However, if the Master **does** implement Modbus/TCP sequence numbers, then the stale responses are rejected. If the Master is smart enough to resynchronize itself (response #B doesn't kill poll #C, but Master waits more), then this resynchronization will manifest itself as the slaves **going off-line and back on-line intermittently.** If the Master is not smart enough to resynchronize, once this out-of-sync behavior occurs, your **slaves go permanently off-line**.

As you can see, this Modbus/TCP master is out of sync and the only cure may be to either restart the Master or power cycle our Modbus Bridge. Both actions close the socket and purge the backlogged messages. Most unfortunately, it's the power cycle of our Bridge that is fastest. This always causes the light bulb to go on "Ah, this Bridge device is at fault!"

Our Network-to-Serial product brings out this shortcoming in Master/Client Modbus/TCP designs, but even a pure MB/TCP to MB/TCP network would suffer from this problem if the poll cycle approached the average response time. Any Modbus/TCP network going through WAN will discover this.

Ideally **all** Modbus/TCP Master applications must implement the sequence number and gracefully handle receipt of stale responses with unexpected sequence numbers. Unfortunately, the Modbus/TCP specification says that this sequence number is optional and can be used by a master to match responses to requests; however it can usually be just left as zero. The Modbus/TCP slave just echoes this back in the response. So most Modbus/TCP OPC servers today do not implement the sequence number.

Fortunately, a second generation of Modbus/TCP masters is starting to come that understands the issues of dealing with a Modbus Bridge to serial. So what is your solution if your Modbus/TCP master is first generation?

- Slow down your poll rate. You have to consider the worst-case response time – assume all polls timeout. If you have 5 slaves that normally answer in less than 100 msec each, but you must use a 250 msec message timeout, then polling each of the 5 each 1.25 sec is the only promised safe rate.
- If you are only polling a single slave (or poll one slave at a time), then you can try the "Disable Pipeline" option in the Modbus Bridge firmware. This will either help or make things hopelessly worse. If your OPC server or host application relies on pipelining to send more than one outstanding poll at once, then disabling the pipeline will essentially stop all data communication. (In which case, you can just turn the pipeline back on!)

The ideal solution (the 2nd generation solution) is for your Modbus/TCP Master/Client to not only support the Sequence Number, but also support the receipt of the 00AH and 00BH extended Modbus/TCP exception response. Then the Master/Client never needs to do retries – for each poll, it **will** receive either a value Modbus/TCP response or a Modbus/TCP exception that the slave is unreachable or timed out. This prevents the Master/Client from sending more polls than the Modbus Bridge can process and building the TCP buffer queue up in the first place.

5. Technical Support

If you are experiencing a problem that is not described in this manual, please contact Advantage-Devices at +49 6151 6294871